# Table of Contents

Document Management ........................................................................................................... 3

1. Overview .......................................................................................................................... 4

2. Design .............................................................................................................................. 5

   2.1 CRC Schema with plug-in specific features highlighted ............................................... 6

   2.2 Tables .......................................................................................................................... 7

      2.2.1 QT_ANALYSIS_PLUGIN ....................................................................................... 7
      2.2.2 QT_PRIVILEGE .................................................................................................. 8
      2.2.3 QT_QUERY_MASTER ............................................................................................ 8
      2.2.4 QT_QUERY_INSTANCE .......................................................................................... 9
      2.2.5 QT_QUERY_RESULT_INSTANCE ............................................................................. 9
      2.2.6 QT_PATIENT_SET_COLLECTION .................................................................... 9
      2.2.7 QT_PATIENT_ENC_COLLECTION ........................................................................ 9
      2.2.8 QT_XML_RESULT .................................................................................................. 9

3. Message ............................................................................................................................ 10

   3.1 Analysis Definition Type ............................................................................................. 10

   3.2 Example Message ........................................................................................................ 11

4. Architecture ...................................................................................................................... 13

   4.1 Plug-in pseudo code ................................................................................................... 14

5. Install ................................................................................................................................. 15
<table>
<thead>
<tr>
<th>Revision Number</th>
<th>Date</th>
<th>Author</th>
<th>Description of change</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.6.1</td>
<td>07/22/10</td>
<td>Janice Donahoe</td>
<td>Created 1.6 version of document.</td>
</tr>
</tbody>
</table>
1. OVERVIEW

The analysis plug-in is one of the ways to extend the CRC cell. The plug-ins are a custom process which can be started from the command line within the CRC cell. The plug-ins can perform anything from simple short tasks like calculating breakdowns of patients by disease, to more time intensive tasks like a full two-set patient comparison analysis which could typically run for almost a day. The plug-ins can be started as a daemon process or can be called from the client via the web service call. The plug-in supports the following main features:

- Supports new analysis process without having to create a new web service.
- Supports scaling by the Queue models
- Tracking the plug-ins run status
- Plug-in authorization based on project and user role
- Auto cleanup of run data
2. DESIGN

The analysis plug-in uses a lot of the same tables that are used by the standard CRC queries.

The requests and results for the plug-in are stored in the existing query tables. As is typical for existing queries, the output goes into any of the three result tables.

- Patient Set Table
- Encounter Set Table
- XML Result table

If the results cannot fit into any of the above result tables:

- It can have its own local tables and have the pointer of the result in the XML result table.
- It can also produce file output that can be accessed through the file repository.
2.1 CRC Schema with plug-in specific features highlighted
2.2 Tables

2.2.1 QT_ANALYSIS_PLUGIN

The plug-in’s metadata for an individual project is stored in this QT_ANALYSIS_PLUGIN table. The CRC looks up the command_line field by the plug_name, version_cd and the group_id field.

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plugin_id</td>
<td>Uniquely generated id and primary key of the table</td>
</tr>
<tr>
<td>group_id</td>
<td>Project_id values goes in this column, and is a project that is registered to run a plug-in. Each project needs a separate row to run a plug-in (and thus can have different run parameters). A plug-in can be run by all projects if this column has a value of “@”</td>
</tr>
<tr>
<td>plugin_name</td>
<td>Plug-in name, consistent across versions of plug-in</td>
</tr>
<tr>
<td>version_cd</td>
<td>Plug-in version code</td>
</tr>
<tr>
<td>description</td>
<td>Description</td>
</tr>
<tr>
<td>command_line</td>
<td>The value of the field will be the full path of the plug-in script for a particular project. By having separate command line per project/group supports constraining the resource allocation. i.e. a project plug-in can be set with some specific memory, priority, etc.</td>
</tr>
<tr>
<td>working_folder</td>
<td>This column specifies the folder name where the command line process will be started.</td>
</tr>
<tr>
<td>status_cd</td>
<td>This column will have the value ‘A’ for active and ‘D’ for deleted.</td>
</tr>
<tr>
<td>commandoptions_cd</td>
<td>Similar to “options” characters that one appends to the Unix command string, the following options are available:</td>
</tr>
<tr>
<td></td>
<td>A: The run instance and result data for the plug-in will be automatically cleaned at the end of the day.</td>
</tr>
<tr>
<td></td>
<td>L: Run in long-running queue only (low priority).</td>
</tr>
<tr>
<td></td>
<td>M: Keep plug-in in memory (for quick startup).</td>
</tr>
<tr>
<td>parent_plugin_id</td>
<td>This field helps to maintain relationship with its parent plug-in</td>
</tr>
<tr>
<td>parameter_info</td>
<td>The value of this column is in the plug-in request xml. The plug-ins can specify their inputs and outputs via the request xml. i.e. analysis_definition_type. The plug-in client can take advantage of this sample xml to render input and output values to users.</td>
</tr>
<tr>
<td>parameter_info_xsd</td>
<td>This column will have the xsd of the parameter_info.</td>
</tr>
</tbody>
</table>
### 2.2.2 QT_PRIVILEGE

The QT_PRIVILEGE table specifies the minimum user role required to access the plug-in.

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>protection_label_cd</td>
<td>This column can be used as follows:</td>
</tr>
<tr>
<td></td>
<td>1. To map the plug-in access control to the user’s role then the value will be ‘@’.</td>
</tr>
<tr>
<td></td>
<td>2. To map access control within the plug-in, then the value could be some marker/label name.</td>
</tr>
<tr>
<td>plugin_id</td>
<td>Plug-in id – negative numbers are used for built-in processes (such as patient-setfinder)</td>
</tr>
<tr>
<td>dataprod_cd</td>
<td>Minimum data track role.</td>
</tr>
<tr>
<td>hivemgmt_Cd</td>
<td>Minimum management track role.</td>
</tr>
</tbody>
</table>

### 2.2.3 QT_QUERY_MASTER

This master table to the holds the client’s analysis plug-in request information. i.e. the user_id, analysis definition, the i2b2 request_xml, etc..

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>query_master_id</td>
<td>Unique generated id for every entry</td>
</tr>
<tr>
<td>name</td>
<td>Name of the plug-in plus the create time stamp.</td>
</tr>
<tr>
<td>master_type_cd</td>
<td>The master_type_cd field determines what type of master record stored. i.e. For the plug-in’s the value would be ‘ANALYSIS_PLUGIN’</td>
</tr>
<tr>
<td>plug_id</td>
<td>Plug-in id - negative numbers are used for built-in processes (such as patient-setfinder)</td>
</tr>
<tr>
<td>request_xml</td>
<td>The analysis definition part of the plug-in run request xml will be stored here.</td>
</tr>
<tr>
<td>i2b2_request_xml</td>
<td>The entire plug-in request which includes the i2b2 header xml is stored here.</td>
</tr>
<tr>
<td>create_date</td>
<td>Create date</td>
</tr>
<tr>
<td>delete_date</td>
<td>Delete date</td>
</tr>
</tbody>
</table>
2.2.4 **QT_QUERY_INSTANCE**

The QT_QUERY_INSTANCE table tracks the plug-in’s execution information. Like start time, status, result, etc.

2.2.5 **QT_QUERY_RESULT_INSTANCE**

The QT_QUERY_RESULT_INSTANCE table holds the pointer to instance and the result information. It has the many to one relation with QT_QUERY_INSTANCE table. There could be more than one result type for a plug-in run instance. Hold status and size for of the plug-in result.

2.2.6 **QT_PATIENT_SET_COLLECTION**

The QT_PATIENT_SET_COLLECTION table holds a list of patient numbers for a result instance id. The set_index field holds index value for the list.

2.2.7 **QT_PATIENT_ENC_COLLECTION**

The QT_PATIENT_ENC_COLLECTION table holds a list of patient numbers and corresponding encounter number (visit) for a result instance id.

2.2.8 **QT_XML_RESULT**

The QT_XML_RESULT table stores the result in XML format. The schema for the common XML result format is in the `i2b2_result_msg.xsd`.
3. MESSAGE

Run an Analysis plug-in by passing the plug-in name and its parameters. The analysis name and parameter usually is part of the individual Analysis plug-in document. The response message for this request is similar to the setfinder’s run query request.

<table>
<thead>
<tr>
<th>Request Type</th>
<th>Request</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRC_QRY_runQueryInstance_fromAnalysisDefinition</td>
<td>analysis_definitionType</td>
<td>master_instance_result_responseType</td>
</tr>
</tbody>
</table>

3.1 Analysis Definition Type

<table>
<thead>
<tr>
<th>Element Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>analysis_plugin_name</td>
<td>Name of the plug-in</td>
</tr>
</tbody>
</table>
| crc_analysis_input_param     | This element contains the input xml that is defined in the parameter_info_xsd column of the qt_analysis_plugin table.  

*Example:*

```xml
<crc_analysis_input_param>
    <param type="int" column="param1">
        param1 values
    </param>
    ...
</crc_analysis_input_param>
```

| crc_analysis_result_list     | This element contains the output xml that is defined in the parameter_info_xsd column of the qt_analysis_plugin table.  

```xml
<crc_analysis_result_list>
    <result_output full_name="XML" priority_index="1" name="XML"/>
    ...
</crc_analysis_result_list>
```
3.2 Example Message

```xml
<request_header>
  <result_waittime_ms>90000</result_waittime_ms>
</request_header>

<message_body>
  <crc:psmheader>
    <request_type>
      CRC_QRY_runQueryInstance_fromAnalysisDefinition
    </request_type>
  </crc:psmheader>

  <crc:request xsi:type="crc:analysis_definition_requestType">
    <analysis_definition>
      <analysis_plugin_name>CALCULATE_PATIENTCOUNT_FROM_CONCEPTPATH</analysis_plugin_name>
      <crc_analysis_input_param name="ONT request">
        <param type="int" column="item_key">
          \rpdr\RPDR\Diagnoses\Circulatory system (390-459)\rdr
        </param>
      </crc_analysis_input_param>
      <crc_analysis_result_list>
        <result_output full_name="XML" priority_index="1" name="XML"/>
      </crc_analysis_result_list>
    </analysis_definition>
  </crc:request>

  <crc:response xsi:type="crc:master_instance_responseType">
    <query_master>
      <query_master_id>0</query_master_id>
      <name>CALCULATE_PATIENTCOUNT_FROM_CONCEPTPATH</name>
      <user_id/>
      <group_id/>
      <create_date>2000-12-30T00:00:00</create_date>
      <request_xml/>
    </query_master>
    <query_instance>
      <query_instance_id>0</query_instance_id>
      <query_master_id>0</query_master_id>
      <user_id/>
      <group_id/>
    </query_instance>
  </crc:response>
```
4. ARCHITECTURE

The CRC plug-ins can be developed in any language as long as it can be packaged as a command line script.

The analysis request is processed first by storing the request information in the QT_QUERY_MASTER table and then the plug-in process will be started using the Apache Exec Utility. The exec process will be started with a set timeout value and will be terminated automatically if the process does not complete before the timeout value.

The client can specify in the analysis request message the maximum timeout value ($<result_waittime_ms>$).

If the plug-in process does not complete within the wait time value, then it will be flagged to run in the queue. There are two types of queues.

1. Medium Queue
   - Handles medium size jobs; maximum time to complete is four hours.
   - If a job can’t complete in this queue, then it is flagged to run in the large queue.

2. Large Queue
   - Supports jobs that can run for any configured maximum length.
   - In order to reduce the load on the server, these queues run jobs serially; at the most two queues are running (also configurable).
4.1 Plug-in pseudo code

CRC will pass parameters to the plug-in's command line script. Using one of the incoming parameters, the plug-in will lookup the datasource/db connection. Then plug-in can then read the database to process the request and finally write the result.

BEGIN
   -- Read the command line argument [-domain_id, -project_id, -user_id, -plugin_id]
   -- Lookup the project datasource using the domain, project and user id,
   -- Using the plugin_id, plug-in can access the analysis plug-in definition which is part of the requests for further process.
   -- Write the results (patient set/encounter set/XML format) to the result table
END;
5. INSTALL

To register a plug-in, simply add the entry in the QT_ANALYSIS_PLUGIN table and set up the plug-in script in appropriate folder.

The following steps show how to install a sample analysis plug-in named “CALCULATE_PATIENTCOUNT_FROM_CONCEPTPATH”

1. Create the analysis service directory and copy the jar files.
   - ant -f analysis_launch.xml deploy

2. Register the analysis plug-in information to the table:
   a. Open the database client and run the following SQL for each project databases.

   ```sql
   insert into QT_ANALYSIS_PLUGIN(plugin_id, plugin_name, description, version, command_line, working_folder, status_id) values ('1','CALCULATE_PATIENTCOUNT_FROM_CONCEPTPATH', 'CALCULATE_PATIENTCOUNT_FROM_CONCEPTPATH', '1.0', '/opt/jboss/server/default/analysis_commons_launcher/bin/run_conceptpatient_breakdown.sh', '/opt/jboss/server/default/analysis_commons_launcher/bin', 'A');
   ```

3. Copy each project’s datasource registered in crc-ds.xml to “CRCApplicationContext.xml”. The bean id is the datasource name and that would start with ‘java:’ plus the datasource name in crc-ds.xml.

   ```xml
   <bean id="java:QueryToolDemoDS" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">
       <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
       <property name="url" value="jdbc:oracle:thin:@localhost:1521:XE"/>
       <property name="username" value="i2b2demodata"/>
       <property name="password" value="i2b2demodata_password"/>
   </bean>
   ```